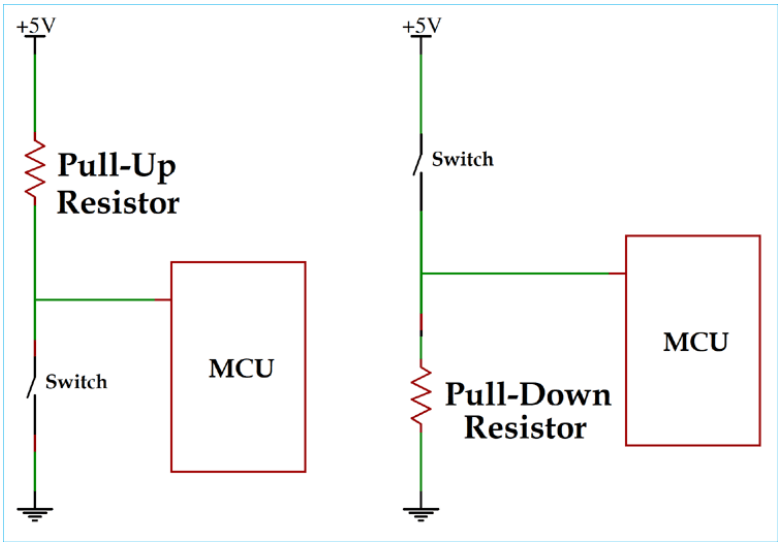
Table of Contents

1 Dokumentation

1.1 Allgemeine Infos

1.1.1 Pull up vs Pull down



1.1.2 Wichtige Import

```
#define F_CPU 16000000  // setzt die Frequenz der CPU zu 16MHz
#include <util/delay.h>
#include <avr/io.h>
#include <avr/interrupt.h> // wird benötigt um Interrupts zu verwenden
#include <util/atomic.h> // wird benötigt um atomic-Blöcke zu nutzen
```

1.1.3 Interrupts aktivieren

```
sei();
```

1.1.4 Atomic Block

```
ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {
    // hierhin kommt der Code mit atomarem Zugriff
}
```

1.1.5 Makro-Definitionen

```
#define _BM(x) (1<<x)
```

1.2 Ports

1.2.1 DDRx

- Setzt den Port als Input, wenn 0 oder als Output, wenn 1

1.2.2 PORTx

- Bei Input Ports wird der interne Pull up mit 1 ein oder mit 0 ausgeschalten
- Bei Output Ports wird der Port auf High oder Low geschalten

1.2.3 Beispiel Konfiguration

```
DDRB |= ((1<<PORTB0) | (1<<PORTB1) | (1<<PORTB2)); // setzt die Pins 0, 1 und 2 auf PortB als Output
PORTB |= (1<<PORTB0); // setzt Pin 0 auf High
DDRC &= ~(1<<PORTC4); // setzt den Pin 4 auf PortC als Input
PORTC &= ~(1<<PORTC4); // deaktiviert den internen Pull Up
```

1.3 Pin Change Interrupts

1.3.1 Wichtige Register

- **PCICR:** legt fest welche Gruppe von PCI aktiviert ist
- **PCIFR:** Flags, die gesetzt werden, wenn ein Interrupt auftritt
- **PCMSK0 / PCMSK1 / PCMSK2:** Masken, um die PCI auf einzelnen Pins zu aktivieren

1.3.2 Interrupt-Vektoren

- **PCINT0_vect:** Pins 0-7
- **PCINT1_vect:** Pins 8-14
- **PCINT2_vect:** Pins 16-23

1.3.3 PCICR

PCICR – Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0	
(0x68)	–	–	–	–	–	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **PCIE0:** Pins 0-7
- **PCIE1:** Pins 8-14
- **PCIE2:** Pins 16-23

1.3.4 PCICR

PCIFR – Pin Change Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1B (0x3B)	–	–	–	–	–	PCIF2	PCIF1	PCIF0	PCIFR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **PCIF0:** Pins 0-7
- **PCIF1:** Pins 8-14
- **PCIF2:** Pins 16-23

1.3.5 PCMSK<0-2>

PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PCMSK1 – Pin Change Mask Register 1

Bit	7	6	5	4	3	2	1	0	
(0x6C)	–	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PCMSK2 – Pin Change Mask Register 2

Bit	7	6	5	4	3	2	1	0	
(0x6D)	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

1.3.6 Beispiel Konfiguration

```
PCMSK1 |= (1 << PCINT8);    // erlaubt Pin 8 auf Port C Interrupts auszulösen
PCMSK1 |= (1 << PCINT9);    // erlaubt Pin 9 auf Port C Interrupts auszulösen
PCMSK1 |= (1 << PCINT10);   // erlaubt Pin 10 auf Port C Interrupts auszulösen
PCICR |= (1 << PCIE1);      // Pin Change Interrupt Change Register (welche Register einen Interrupt auslösend darf)
```

```
ISR(PCINT1_vect)
{
    if(value)
    {
        value = 0;
    }
    else
    {
        value = 1;
    }
}
```

1.4 ADC

1.4.1 Wichtige Register

- **ADMUX:** legt die Referenzspannung fest und dein Eingangspin
- **ADCSRA:** setzen von Interrupt, Prescaler, starten einer Konvertierung
- **ADCSRB:** um im Auto-Trigger-Mode die Trigger-Quelle zu selektieren
- **ADCW:** enthält das Ergebnis der Konvertierung

1.4.2 Interrupt-Vektoren

- **ADC_vect:** wenn eine Konvertierung abgeschlossen ist

1.4.3 ADMUX

ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, internal V_{REF} turned off
0	1	AV_{CC} with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 1.1V voltage reference with external capacitor at AREF pin

Table 23-4. Input Channel Selections

MUX3..0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	ADC8[1]
1001	(reserved)
1010	(reserved)
1011	(reserved)
1100	(reserved)
1101	(reserved)
1110	1.1V (V_{BG})
1111	0V (GND)

1.4.4 ADCSRA

ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **ADEN:** aktiviert den ADC (schaltet ihn ein, wenn 1 oder aus, wenn 0)
- **ADSC:** um eine einzige Konvertierung zu starten
- **ADATE:** um Auto Trigger zu aktivieren (um die Konvertierung automatisch zu starten)
- **ADIF:** wenn eine Konvertierung abgeschlossen ist, wird dieses Flag auf 1 gesetzt
- **ADIE:** um den ADC-Interrupt zu aktivieren
- **ADPS:** um eine Prescaler einzustellen dieser legt die Sampling Rate des ADCs fest
    - Beim verbauten ADC sollte diese zwischen 50kHz und 200kHz liegen → bei 16MHz ist ein **Prescaler von 128** zu wählen

Table 23-5. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

1.4.5 ADCSRB

ADCSRB – ADC Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
(0x7B)	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 23-6. ADC Auto Trigger Source Selections

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free running mode
0	0	1	Analog comparator
0	1	0	External interrupt request 0
0	1	1	Timer/Counter0 compare match A
1	0	0	Timer/Counter0 overflow
1	0	1	Timer/Counter1 compare match B
1	1	0	Timer/Counter1 overflow
1	1	1	Timer/Counter1 capture event

1.4.6 ADCW

- Bei dem Wert handelt es sich um eine nicht um einen dezimalen Volt-Wert
- Es handelt sich um eine 10bit-ADC → Wertebereich 0…1023
- Formel zur Berechnung der Spannung:
  - $Spannung = \frac{ADCW}{1024} * MaxSpannung$

1.4.7 Beispiel Konfiguration

```
DDRC &= ~(1<<PORTC5); // Port wird auf Input gesetzt
```

```
ADMUX |= (1<<REFS0); // AV mit externem capcitor auf AREF pin
ADMUX |= (1<<MUX0) | (1<<MUX2); // ADC Pin 5 wird aktiviert (auf PortC Pin 5)
ADCSRA |= (1<<ADEN); // aktiviert den ADC
ADCSRA |= (1<<ADPS0) | (1<<ADPS1) | (1<<ADPS2); // setzt den Prescaler auf 128
ADCSRA |= (1<<ADIE); // aktiviert ADC Interrupts
ADCSRA |= (1<<ADSC); // startet die Konvertierung
```

```
ISR(ADC_vect)
{
    adcResult=ADCW; // ADCW enthält das Ergebnis des ADCs
    voltage=(adcResult/1024.0)*5.0; // Convertierung zu Volt ((ADCW / Resolution [10-bit = 1024]) * maximale Volt [5V])
    ADCSRA |= (1<<ADSC); // startet die Konvertierung
}
```

1.5 LCD

1.5.1 Ports definieren (lcd_definitions.h)

```
#define LCD_PORT         PORTD        /**< port for the LCD lines   */
#define LCD_PORT1        PORTC
#define LCD_DATA0_PORT   LCD_PORT     /**< port for 4bit data bit 4 */
#define LCD_DATA1_PORT   LCD_PORT     /**< port for 4bit data bit 5 */
#define LCD_DATA2_PORT   LCD_PORT     /**< port for 4bit data bit 6 */
#define LCD_DATA3_PORT   LCD_PORT     /**< port for 4bit data bit 7 */
#define LCD_DATA0_PIN    2            /**< pin for 4bit data bit 4  */
#define LCD_DATA1_PIN    3            /**< pin for 4bit data bit 5  */
#define LCD_DATA2_PIN    4            /**< pin for 4bit data bit 6  */
#define LCD_DATA3_PIN    5            /**< pin for 4bit data bit 7  */
#define LCD_RS_PORT      LCD_PORT1     /**< port for RS line         */
#define LCD_RS_PIN       0            /**< pin  for RS line         */
#define LCD_RW_PORT      LCD_PORT1     /**< port for RW line         */
#define LCD_RW_PIN       1            /**< pin  for RW line         */
#define LCD_E_PORT       LCD_PORT1     /**< port for Enable line     */
#define LCD_E_PIN        2            /**< pin  for Enable line     */
```

1.5.2 Library konfigurieren (lcd.h)

```
#define _LCD_DEFINITIONS_FILE
```

- Dabei wird die Nutzung des lcd_definitions.h Files festgelegt

1.5.3 Import

```
#include "lcd.h"
```

1.5.4 Initialisieren

```
lcd_init(LCD_DISP_ON); // das Display wird initialisiert
```

1.5.5 Verwendung

```
char str[16];  // string für die Spannungsausgabe

lcd_clrscr(); // löschen des Displays

lcd_gotoxy(0, 0); // setzen der Cursor Position

sprintf(str, "%d", direction); // Formatierung der Richtung
lcd_puts(str); // Ausgabe der Richtung

lcd_gotoxy(0, 1);
dtostrf(percentage, 6, 2, str); // float to string
sprintf(str, "%s %%", str); // Formatierung des Strings
lcd_puts(str); // Ausgabe der Prozent

_delay_ms(100);
```

1.6 Timer

1.6.1 Timer

- **Timer 0:** ist ein 8-bit Timer
- **Timer 1:** ist ein 16-bit Timer
- **Timer 2:** ist ein 8-bit Timer

1.6.2 Wichtige Register

- **TCCR0A / TCCR1A / TCCR2A:** Verhalten des Timers eintellen
- **TCCR0B / TCCR1B / TCCR2B:** Prescaler setzen
- **TIMSK0 / TIMSK1 / TIMSK2:** um Timer-Interrupts zu aktivieren
- **OCR0A / OCR1A / OCR2A:** um den Compare-Match-Wert zu setzen für Pin OC0A, OC1A oder OC2A
- **OCR0B / OCR1B / OCR2B:** um den Compare-Match-Wert zu setzen für Pin OC0B, OC1B oder OC2B

1.6.3 Interrupt-Vektoren

- **TIMER0_COMPA_vect / TIMER1_COMPA_vect / TIMER2_COMPA_vect:** wenn der Wert dem von OCA0, OCA1 oder OCA2 entspricht
- **TIMER0_COMPB_vect / TIMER1_COMPB_vect / TIMER2_COMPB_vect:** wenn der Wert dem von OCB0, OCB1 oder OCA2 entspricht

1.6.4 TCCR0A

TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 14-2. Compare Output Mode, non-PWM Mode

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Toggle OC0A on compare match
1	0	Clear OC0A on compare match
1	1	Set OC0A on compare match

Table 14-5. Compare Output Mode, non-PWM Mode

COM0B1	COM0B0	Description
0	0	Normal port operation, OC0B disconnected.
0	1	Toggle OC0B on compare match
1	0	Clear OC0B on compare match
1	1	Set OC0B on compare match

Table 14-8. Waveform Generation Mode Bit Description

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on(1)(2)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, phase correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, phase correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

1.6.5 TCCR0B

TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	**WGM02**	**CS02**	**CS01**	**CS00**	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 14-9. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)
0	1	1	$clk_{I/O}$/64 (from prescaler)
1	0	0	$clk_{I/O}$/256 (from prescaler)
1	0	1	$clk_{I/O}$/1024 (from prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

- Prescaler Berechnung:
  - $Zeit = \frac{Register\ Größe\ (256)}{16\,000\,000} * Prescaler$
  - $Prescaler = \frac{16\,000\,000 * Zeit}{Schritte}$

1.6.6 TIMSK0

TIMSK0 – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	**OCIE0B**	**OCIE0A**	**TOIE0**	**TIMSK0**
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **OCIE0A:** Compare-Match für OC0A
- **OCIE0B:** Compare-Match für OC0B
- **TOIE0:** Interrupt, wenn Timer überläuft (Overflow)

1.6.7 Beispiel Konfiguration

```
TCCR0A |= (1 << WGM01);               // setzt den Timer Compare Mode (CTC Mode)
OCR0A = 250;                          // maximaler Wert (bei Takzyklus 64 / 16 000 000 = 4µs => 250 Takte zu je 4µs = 1ms)
TIMSK0 |= (1 << OCIE0A);              // Timerinterrupt aktivieren
TCCR0B |= (1 << CS00) | (1 << CS01);  // setzen des Frequenzteilers CS00 + CS01 = Prescaler von 64
```

```
ISR(TIMER1_COMPA_vect)  // wird aufgerufen, wenn der Timer Compare Interrupt ausgelöst wird
{
    if (state == 1)
    {
        timer++;
    }
}
```

1.7 PWM

1.7.1 Siehe Timer

- **OC0A** und dergleichen werden nun zum Setzen des Duty Cycles verwendet
- **OC0B** und dergleichen werden nun zum Setzen des Duty Cycles verwendet

1.7.2 TCCR0A

Table 14-3. Compare Output Mode, Fast PWM Mode(1)

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	WGM02 = 0: Normal port operation, OC0A disconnected. WGM02 = 1: Toggle OC0A on compare match.
1	0	Clear OC0A on compare match, set OC0A at BOTTOM, (non-inverting mode).
1	1	Set OC0A on compare match, clear OC0A at BOTTOM, (inverting mode).

Table 14-4. Compare Output Mode, Phase Correct PWM Mode(1)

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	WGM02 = 0: Normal port operation, OC0A disconnected. WGM02 = 1: Toggle OC0A on compare match.
1	0	Clear OC0A on compare match when up-counting. Set OC0A on compare match when down-counting.
1	1	Set OC0A on compare match when up-counting. Clear OC0A on compare match when down-counting.

1.7.3 TCCR0B

Table 14-6. Compare Output Mode, Fast PWM Mode(1)

COM0B1	COM0B0	Description
0	0	Normal port operation, OC0B disconnected.
0	1	Reserved
1	0	Clear OC0B on compare match, set OC0B at BOTTOM, (non-inverting mode)
1	1	Set OC0B on compare match, clear OC0B at BOTTOM, (inverting mode).

Table 14-7. Compare Output Mode, Phase Correct PWM Mode(1)

COM0B1	COM0B0	Description
0	0	Normal port operation, OC0B disconnected.
0	1	Reserved
1	0	Clear OC0B on compare match when up-counting. Set OC0B on compare match when down-counting.
1	1	Set OC0B on compare match when up-counting. Clear OC0B on compare match when down-counting.

1.7.4 Beispiel Konfiguration

```
TCCR1A |= (1<<WGM10) | (1<<WGM12); // setzt den Modus auf Fast PWM
TCCR1A |= (1<<COM1B1); // setzt den Output auf OC1B
TCCR1B |= (1<<CS10); // setzen des Prescalers auf 0
OCR1B = 0xFF;
```

```
int sub = (int)(255.0-255.0/100.0*percentage); // kalkuliert den Wert, der von 0xFF abgezogen wird
OCR1B = 0xFF-sub; // PWM Duty Cycle wird gesetzt (Motorgeschwindigkeit)
```

1.8 Serial Communication

1.8.1 Wichtige Register

- **UCSR0A:** wichtige Flags für UART im Polling Betrieb
- **UCSR0B:** UART-Interrupts
- **UCSR0C:** UART-Settings
- **UDR0:** UART-Daten-Register

1.8.2 Interrupt-Vektoren

- **USART_RX_vect:** UART received data
- **USART_UDRE_vect:** das UART-Datenregister ist wieder frei

1.8.3 UCSR0A

UCSRnA – USART Control and Status Register n A

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- **RXC0:** Flag welches gesetzt wird, wenn eine Nachricht empfangen wurde
- **TXC0:** Flag welches gesetzt wird, wenn das Senden abgeschlossen ist
- **UDRE0:** Flag wird gesetzt, wenn das Datenregister leer ist

1.8.4 UCSR0B

UCSRnB – USART Control and Status Register n B

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **RXCIE0:** wird ausgelöst, wenn etwas empfangen wurde
- **TXCIE0:** wird ausgelöst, wenn etwas gesendet wurde
- **UDRIE:** wird ausgelöst, wenn das UART-Datenregister wieder leer ist
- **RCEN0:** um das Empfangen via UART zu aktivieren
- **TXEN0:** um das Senden via UART zu aktivieren
- **USCZ02:** siehe UCSR0C

1.8.5 UCSR0C

UCSRnC – USART Control and Status Register n C

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

Table 19-4. UMSELn Bits Settings

UMSELn1	UMSELn0	Mode
0	0	Asynchronous USART
0	1	Synchronous USART
1	0	(Reserved)
1	1	Master SPI (MSPIM)(1)

Table 19-5. UPMn Bits Settings

UPMn1	UPMn0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, even parity
1	1	Enabled, odd parity

Table 19-6. USBS Bit Settings

USBSn	Stop Bit(s)
0	1-bit
1	2-bit

Table 19-7. UCSZn Bits Settings

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

Table 19-8. UCPOLn Bit Settings

UCPOLn	Transmitted Data Changed (Output of TxDn Pin)	Received Data Sampled (Input on RxDn Pin)
0	Rising XCKn edge	Falling XCKn edge
1	Falling XCKn edge	Rising XCKn edge

1.8.6 Beispiel Konfiguration Polling

```
#define Baudrate 9600
#define Prescaler ((F_CPU/16/Baudrate) - 1)


void uart_init(void)
{
    UBRR0H |= (Prescaler>>8); // (MSB) sets the
    UBRR0L |= Prescaler; // (LSB) Baudrate to 9600

    UCSR0C |= (1<<UPM01); // aktiviert Even Parity Bit
    UCSR0C |= (1<<UCSZ00) | (1<<UCSZ01); // 8 bit can be sent/received per frame

    UCSR0B |= (1<<RXEN0); // enable serial read
    UCSR0B |= (1<<TXEN0); // enable serial write
}
```

```
char uart_read(void)
{
    while ((UCSR0A & (1 << RXC0)) == 0); // waits until sth. is received
    return UDR0; // reads from the serial register
}
```

```
void uart_write(char *c)
{
    while ((UCSR0A & (1 << UDRE0)) == 0); // waits until sending is possible
    UDR0 = *c; // sets the serial register
}
```

```
void uart_send_string(char *s)
{
    while(*s!='\0') // while pointer hasn't reached the end of the string
    {
        uart_write(s); // write char
        s++; // go to the next element of the array (string)
    }
}
```

1.8.7 Beispiel Konfiguration Interrupts

```
void uartInit()
{
    sei();
    ringbufferInit(&rx);
    ringbufferInit(&tx);

    UBRR0H |= (Prescaler>>8); // (MSB) sets the
    UBRR0L |= Prescaler; // (LSB) Baudrate to 9600

    UCSR0C |= (1<<UPM01); // aktiviert Even Parity Bit
    UCSR0C |= (1<<UCSZ00) | (1<<UCSZ01); // 8 bit can be sent/received per frame

    UCSR0B |= (1<<RXEN0); // enable serial read
    UCSR0B |= (1<<TXEN0); // enable serial write

    UCSR0B |= (1<<RXCIE0); // enabel rx interrupt
    UCSR0B |= (1<<TXCIE0); // enable tx interrupt
}
```

```
int uartSendChar(uint8_t data)
{
    uint32_t state;
    state = ringbufferPut(&tx, data);
    if(state & (1<<RIBU_DATALOST) || !(state & (1<<RIBU_OK)))
    {
        // Error
        return -1;
    } else
    {
        // alles ist gut
        UCSR0B |= (1<<UDRIE0); // enable tx interrupt
        return 0;
    }
}
```

```
int uartGetChar(uint8_t *data)
{
    uint32_t state;
    state = ringbufferGet(&rx, data);
    if(state & (1<<RIBU_EMPTY) || !(state & (1<<RIBU_OK)))
    {
        // Error
        return -1;
    } else
    {
        // alles ist gut
        return 0;
    }
}
```

```
ISR(USART_RX_vect)
{
    ringbufferPut(&rx, UDR0);
    uartSendChar(UDR0);
}
```

```
ISR(USART_UDRE_vect)
{
    uint8_t data;
    uint32_t state;
    state = ringbufferGet(&tx, &data);
    if(state & (1<<RIBU_EMPTY) || !(state & (1<<RIBU_OK)))
    {
        // Error
        UCSR0B &= ~(1<<UDRIE0); // disable tx interrupt
    } else
    {
        // alles ist gut
        UDR0 = data;
    }
}
```

1.9 SPI – Serial Peripheral Interface

```
#define SS_PORT      PORTB
#define SS_DDR       DDRB
#define SS_PIN       PORTB2

#define MOSI_PIN     PORTB3
#define MISO_PIN     PORTB4
#define SCK_PIN      PORTB5

void spi_init();

void SPI_MasterTransmit(uint8_t *value);
```

1.9.1 SPCR

SPCR – SPI Control Register

Bit	7	6	5	4	3	2	1	0	
0x2C (0x4C)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **SPIE:** Interrupt aktivieren
- **SPE:** SPI aktivieren
- **MSTR:** Master/Slave Select, 1 = Master SPI Mode, 0 = Slave SPI Mode

Table 18-5. Relationship Between SCK and the Oscillator Frequency

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

Clock Rate setzen

MOSI : serielle Daten-Eingabe (SDI), Controller sendet an Peripherie

MISO : serielle Daten-Ausgabe (SDO), Peripherie sendet an Controller

SCK : serielle Uhr

SS : Slave Select, mit welchem Gerät, Controller kommuniziert

```
void spi_init()
{
	// MOSI, SCK, und SS ports auf Output setzen
	DDRB |= (1 << MOSI_PIN) | (1 << SCK_PIN) | (1 << SS_PIN);

	// SPI aktivieren, set as master, Clock Rate fosc/16 setzen
	SPCR = (1<<SPIE) | (1<<SPE) | (1<<MSTR) | (1<<SPR0);
}
```

```
void SPI_MasterTransmit(uint8_t *value)
{
	// Enable slave (active low)
	SS_PORT &= ~(1 << SS_PIN);

	/* Start transmission */
	SPDR = *value;

	/* Wait for transmission complete */
	while(!(SPSR & (1<<SPIF)));

	// Disable slave
	SS_PORT |= (1 << SS_PIN);
}
```

```
uint8_t shiftvalue = 0;
for (int i=0; i<led; i++)
{
	shiftvalue |= (1<<i);
}

SPI_MasterTransmit(&shiftvalue);
```

2 Links

2.1 ADC

AVR-Tutorial: ADC – Mikrocontroller.net